LIBRARY OF THE
UNIVERSITY OF ILLINOIS
AT URBANA-CHAMPAIGN
510.84
Il6r
no.160-170
cop.2

The person charging this material is responsible for its return to the library from which it was withdrawn on or before the **Latest Date** stamped below.

Theft, mutilation, and underlining of books are reasons for disciplinary action and may result in dismissal from the University.

UNIVERSITY OF ILLINOIS LIBRARY AT URBANA-CHAMPAIGN

NOV 15 1973

L161—O-1096

Digitized by the Internet Archive
in 2013

http://archive.org/details/memorysearchalgo165lyon

510.84
I16r
no.165
cop. 3

COO-1018-1208

DIGITAL COMPUTER LABORATORY

UNIVERSITY OF ILLINOIS

URBANA, ILLINOIS

REPORT NO. 165

MEMORY SEARCH ALGORITHMS FOR AN ITERATIVE PROCESSOR

by

Stephen Mark Lyons

July 17, 1964

(This work is being submitted in partial fulfillment of the requirements for the degree of Master of Science.)

cop. 3

ACKNOWLEDGMENT

The author wishes to thank Dr. Gernot Metze for his supervision of the preparation of this paper, and for his patience. He also gratefully acknowledges the invaluable assistance given by Dr. S. R. Ray.

TABLE OF CONTENTS

		Page
SUMMARY		v
CHAPTER I.	INTRODUCTION TO THE INVESTIGATION.	1
CHAPTER II.	DISCUSSION OF THE TRANSFER MEMORY AND THE ITERATIVE ARRAY.	3
CHAPTER III.	DERIVATION AND DISCUSSION OF THE SEARCH ALGORITHMS	8
CHAPTER IV.	CONCLUSIONS.	28
BIBLIOGRAPHY		30
APPENDIX		31

SUMMARY

Three search algorithms are derived for the iterative processor of the Illinois Pattern Recognition Computer: a between limits search, a minimum search and an ordered retrieval. The processor is compared to another system in its ability to perform ordered retrieval.

These algorithms represent an investigation into possible uses for the logic existing in the iterative processor not previously explored.

CHAPTER I

INTRODUCTION TO THE INVESTIGATION

A memory search is an operation conducted on all or part of the contents of a memory in order to determine the information contained in the memory. To execute an effective memory search it is necessary to process the memory content with a parallel operation. This can be accomplished with an iterative processor, which is a highly repetitive group of identical modules designed to operate in parallel.

This paper is concerned with the construction of three memory search algorithms for the iterative processor of the Illinois Pattern Recognition Computer [Illiac III]: a between limits search, a minimum search (and its dual, maximum search) and an ordered retrieval. The computer is designed specifically for visual pattern processing [1]* but the extensive logic built into the iterative processor warrants this investigation into other possible uses for the processor. The iterative processor or "Iterative Array" is part of the Pattern Articulation Unit of the Illiac III computer (Fig. 1). Connected to the processor is a small matrix memroy called the Transfer Memory. This is the memory on which the searches will be conducted. The memory searches investigated are not chosen specifically to take advantage of certain properties of the memory or the processor but on the basis of their recognized usefulness.

A between limits search (a search for all memory words between prescribed upper and lower arithmetic bounds) has many applications, such as information retrieval and visual pattern processing. In information retrieval

---

* Numbers in brackets refer to entries in the Bibliography.

this search is needed where the upper and lower arithmetic bounds on words in the memory correspond to coded information in a continuous, bounded range. For example, if all information in the memory concerned Transactions of the IEEE, all binary numbers between 10 and 100 might correspond to issues of the Transactions of the P.G.E.C. Thus, a search for all words with arithmetic value between the limits 10 and 100 would, perhaps, yield information concerning the location of the PGEC issues. In visual pattern processing the between limits search is helpful in data reduction [1].

The minimum search is a search for the arithmetic minimum of the words contained in memory. The search derived in this paper is applicable to words of any list length and is meant as an extension of the machine's ability to perform such a search. A minimum search already exists for three bit binary numbers [2]. However, the approach to the problem was necessarily quite different in the case of longer binary words.

An extension of the minimum search yields a search by which all words are retrieved from memory in order corresponding to their arithmetic value--an ordered retrieval. Due to the current interest in ordered retrieval, caused by the application to business and information files [3], a study is made of the adaptation of the iterative processor to this search. The results of this investigation are then compared to a system designed specifically for ordered retrieval [3].

The iterative processor and associated memory are discussed in Chapter II. The three search algorithms are derived and explored in detail in Chapter III. In the last chapter, Chapter IV, concluding remarks are made and possible extensions of the investigation are discussed.

CHAPTER II

DISCUSSION OF THE TRANSFER MEMORY AND THE ITERATIVE ARRAY

The Transfer Memory [Figs. 1,2] is a random access, linear select core memory of 48 x 1024 bits arranged for read or write operations either by long, 1024 bit words (lying in the plane orthogonal to the Z axis) or short 48 bit (Z axis) words. [Fig. 1]

Logically connected to the Transfer Memory is an iterative processor called the Iterative Array [Fig. 3]. The Iterative Array is designed to process the long, 1024 bit, plane words parallel by word. That is, considering the Z axis words, it is able to process, simultaneously, the same bit position of all 1024 Z axis (short) words in parallel.

This 1024 bit parallel processor is composed of nine horizontal registers labeled M and S_1 through S_8 in Fig. 3 (shown for one module). These registers are interconnected to provide logical functions of vertically-related cells (i.e., bits of the same Z words). The output to and the input from the eight neighbor modules is provided for pattern recognition processes. Since the nearest neighbor logic is not germane to the development of this paper, it will not be discussed. Note also that all data passing between the Iterative Array and the Transfer Memory must be transferred via the M register [Fig. 2].

By controlling the gating of the OUTBUS [Fig. 3] the following functions of nine variables or any subset of the variables can be formed on the OUTBUS:

$$f_1 = M \cdot S_1 \cdot S_2 \ldots S_8 \qquad \cdot \text{ means logical AND}$$

$$f_2 = \overline{f}_1$$

$f_3 = M \vee S_1 \vee S_2 \cdots \vee S_8$ $\vee$ means logical OR

$f_4 = \bar{f}_3$

Either of two functions* formed on the INBUS may be gated into one or more of the registers:

$$g_1 = M \vee \left(\bigvee_{i=0}^{8} \text{OUTBUS}_i\right)$$

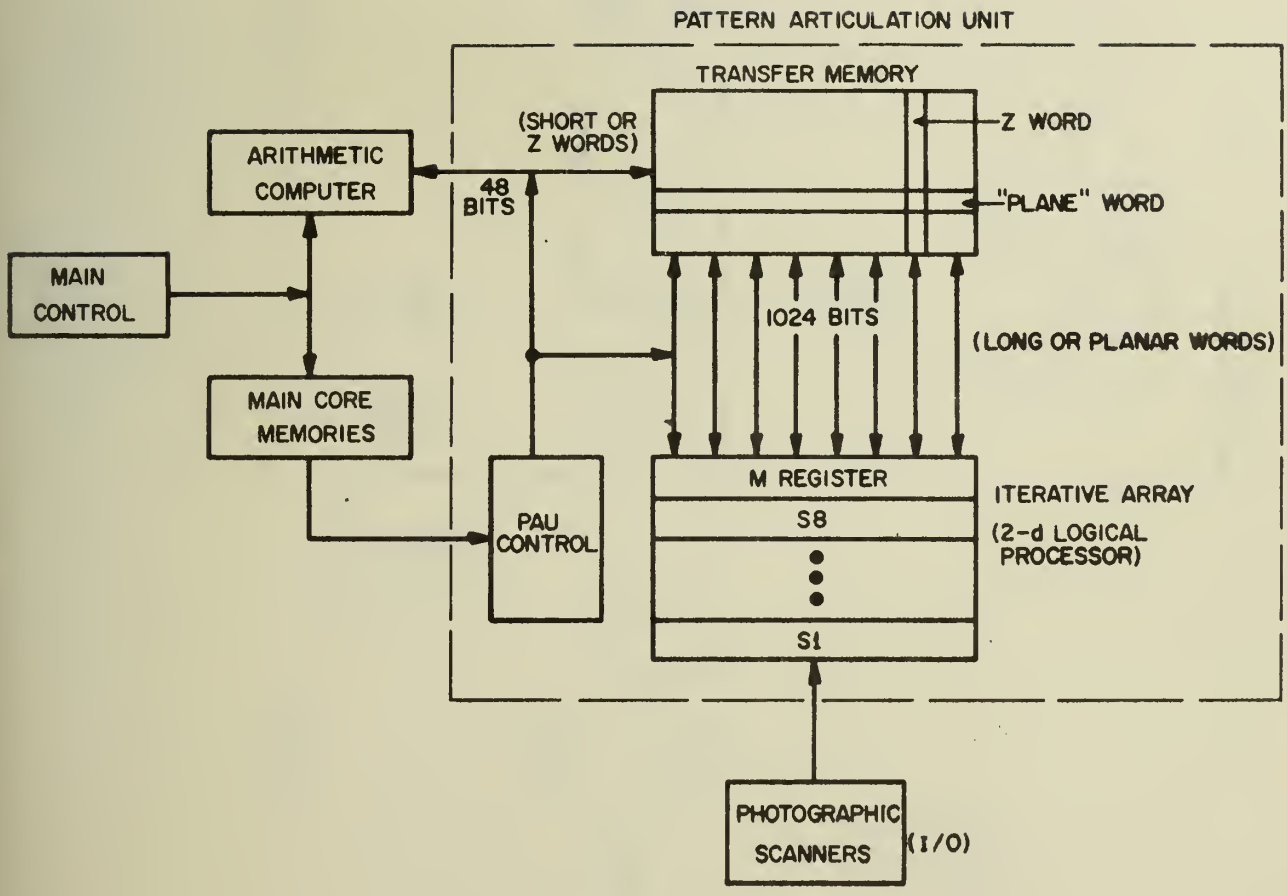
$g_2 = \bar{g}_1$

There is one restriction on the input, however. A function cannot be gated into a register that is also an argument of the function. As an example: $S_8 \vee S_7$ could not be an input to either register S_7 or S_8.

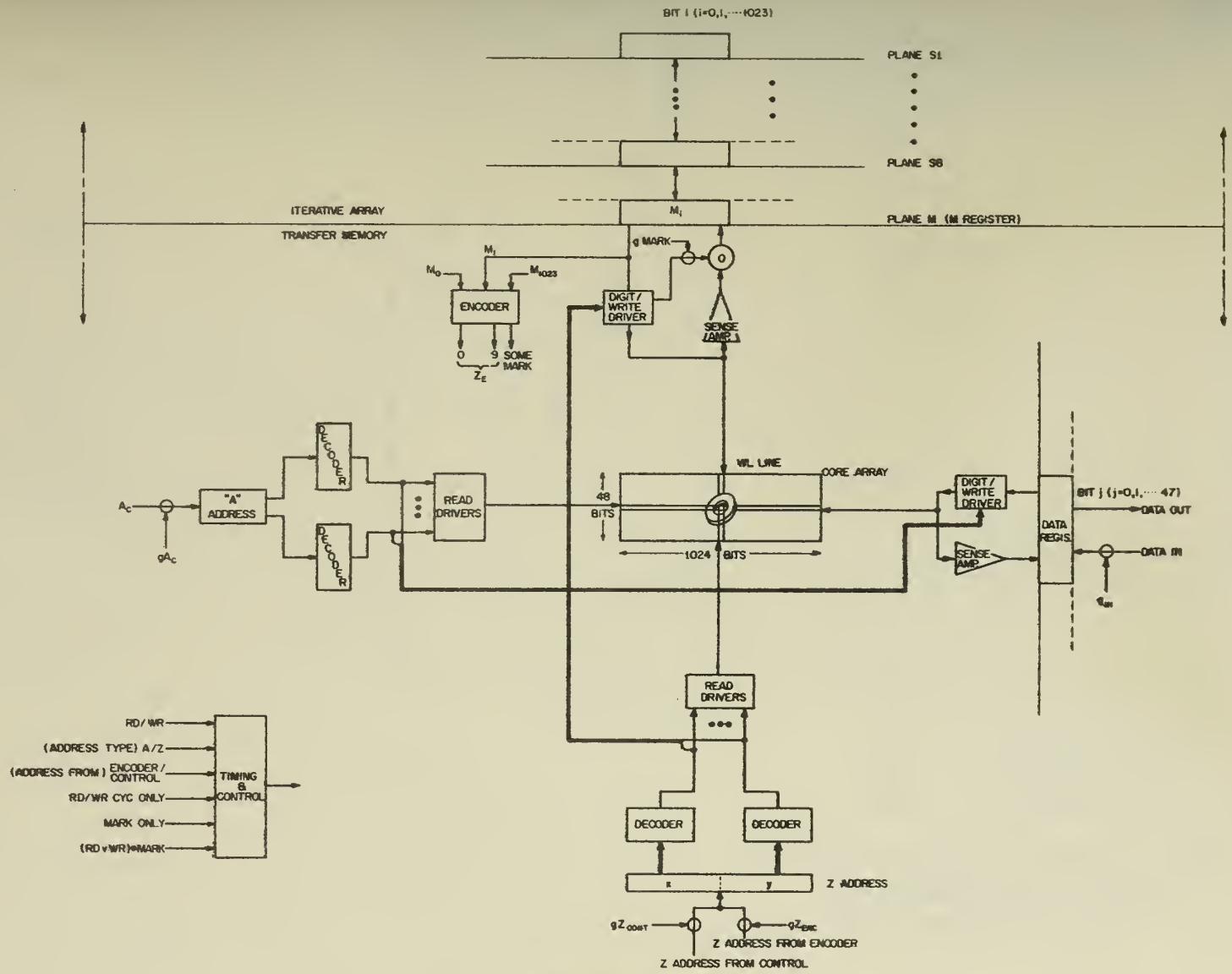
Connected to the M register is an encoder [Fig. 2]. It consists of combinational circuitry which delivers either a ten-bit binary number equal to the minimum coordinate (bit number) of the M register which contains "0", or a one-bit signal ("SOME MARK") indicating the existence of at least one "0" in the M register. Zeros may be interpreted as interrogation responses as will be shown. After the encoder has addressed a position of the M register containing a zero, that zero may be reset to one by means of a MARK operation which then allows the encoder to proceed to the next zero.

---

* The formation of any INBUS or OUTBUS function takes 0.5 microsecond. (See [2].)

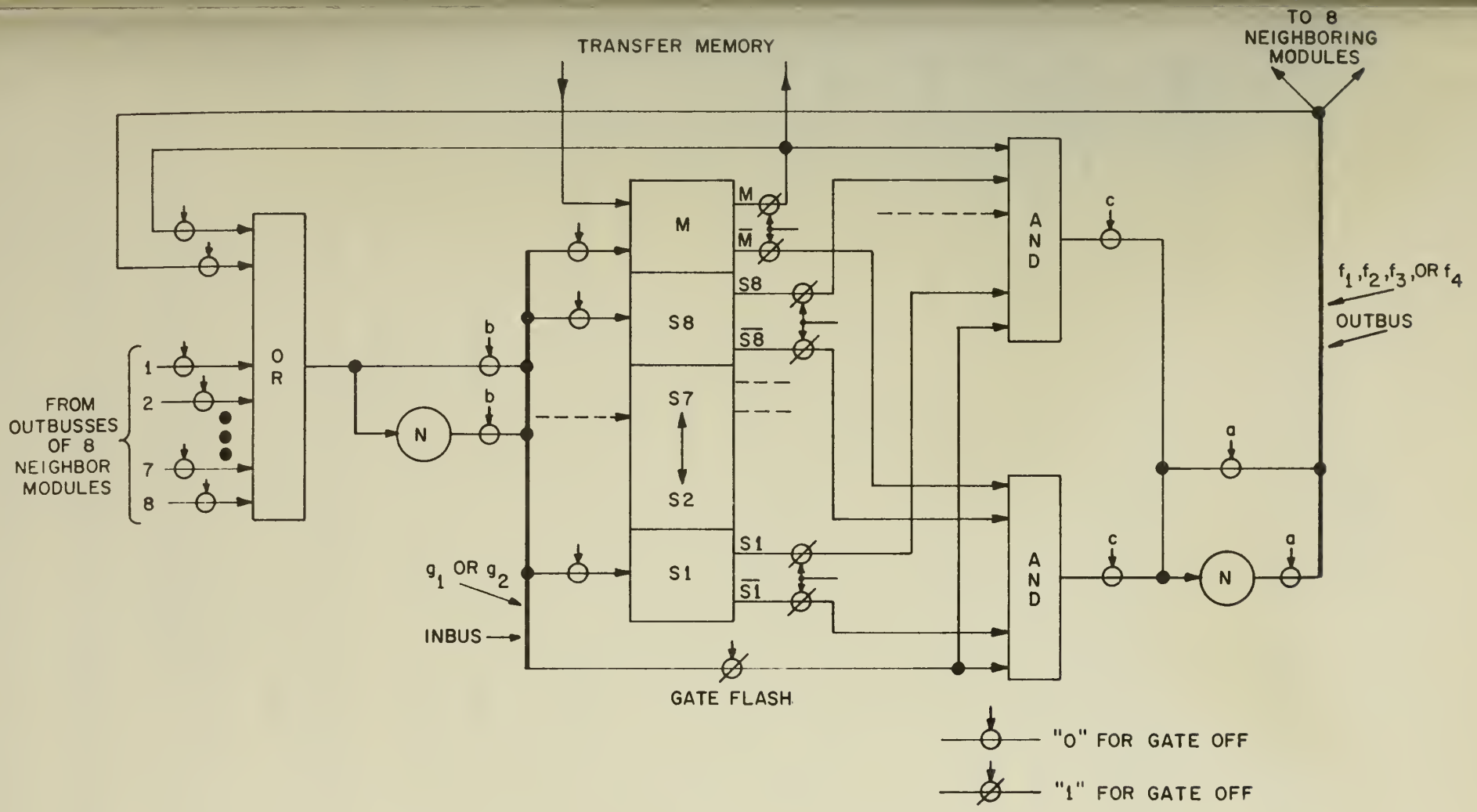


ILLIAC III SYSTEM
FIG. 1



TRANSFER MEMORY ORGANIZATION

FIG. 2



LOGIC FOR ONE MODULE OF ITERATIVE ARRAY

FIG. 3

CHAPTER III

DERIVATION AND DISCUSSION OF THE SEARCH ALGORITHMS

Between Limits Search

An interrogation of the Transfer Memory by the Iterative Array may be performed to find all words in the memory between specified upper and lower arithmetic bounds. The words describing the upper and lower arithmetic bounds are contained in the arithmetic computer [Fig. 1] where they are processed to determine the search executed by the Iterative Array.

A between limits search of the memory contents for all words[1] K such that $A \geq K \geq B$, for any two words A and B, will be constructed in two parts. First an algorithm is formed to obtain all words $K \geq B$. This information will be stored and used as a mask for the second search--that of finding all words $K \leq A$. That is, the second search is executed only on those words passing the first search. An objective in the construction of such algorithms is that the Transfer Memory not be used to store intermediate information obtained in the search, because of the time factor involved in going from the Iterative Array to the Transfer Memory.[2] For this reason registers S_8 and S_7 are used to store information derived from the logical processing of information contained in the other registers.

The rows[3] of the Transfer Memory will be transferred in groups of six to the Iterative Array, starting at the most significant end (the end of

---

[1] A word will always refer to one of the 1024, short 48 bit ("Z axis") words in the Transfer Memory. However, any search may be conducted on only a specific part (or "Key") of the Z axis words.

[2] A transfer of a word between the Iterative Array and the Transfer Memory is performed in 2μsec--four times as long as a formation of a function in the Iterative Array.

[3] A row consists of 1024 bits, one bit from the same bit position of each of the Z axis words. A row is, therefore, one of the previously mentioned "long" words.

greatest arithmetic value) of the words stored. Let the i th row be designated R_i. The word B,

$$B = b_1\ b_2\ b_3 \cdots b_i \cdots b_n,$$

is made available to the arithmetic computer. The processing of the rows in memory by the Iterative Array is determined by the bit value of B in the following manner:

(1) If $b_i = 1$

$$(S_7 \vee R_i)\ S_8 \longrightarrow M \longrightarrow S_8$$

(2) If $b_i = 0$

$$(R_i)\ S_8 \vee S_7 \longrightarrow M \longrightarrow S_7$$

(3) Repeat steps (1) and (2) for all remaining rows incrementing i on each pass.

(4) At the end of the operation

$$\overline{S}_8 \longrightarrow S_7 \qquad \text{Set } S_8 \text{ to 1.}$$

All words greater than or equal to B correspond to zeros in S_7.

Now the word A,

$$A = a_1\ a_2\ a_3 \cdots a_i \cdots a_n,$$

is made available to the arithmetic computer so that a search can be conducted on all words corresponding to zeros in S_7 to find those less than or equal to A. The search is accomplished with the following algorithm:

(1) If $a_i = 0$

$$S_8(S_7 \vee R_i) \longrightarrow M \longrightarrow S_7$$

(2) If $a_i = 1$

$$(R_i)\ S_8 \vee S_7 \longrightarrow M \longrightarrow S_8$$

(3) Repeat steps (1) and (2) for all remaining rows incrementing i on each pass.

(4) $S_7 \longrightarrow M$

The words indicated by zeros in M at step (4) are the words that lie between the limits A and B.

The algorithms used in each part of the search are duals so an analysis of one will be sufficient. Consider the search for $K \geq B$. Register S_7 is designated to store ones corresponding only to words that have been found to be greater than B and register S_8 to store zeros corresponding only to those words previously determined to be less than B. If b_i is equal to one, then no information is obtained as to words greater than B but any word having a zero in this position, that has not previously been determined greater than B, has now failed. This information must be stored in S_8. To propagate ones in positions of words previously determined greater than B, the logical sum of the row being tested and S_7 is taken. If the result derived from this OR operation and S_8 are combined in a logical product, only zeros corresponding to failures are formed [step (1)]. When b_i is equal to zero all ones in the ith row, corresponding to words that have not previously failed, are greater than B. To propagate failures, the logical AND of this row and S_8 is taken. The result of this logical product and register S_7 are combined in a logical sum to give ones in the positions, in S_7, corresponding to words greater than B--the desired result.

To show that a word greater than B is always recorded correctly, assume that a zero is recorded in S_8 corresponding to a word greater than B. This could only happen at step (1) which implies that the word had not previously passed (or else a one would have been recorded in S_7, corresponding to its position) and that when $b_i = 1$, there is a zero in the words ith position. Therefore, the word would be less than B--a contradiction.

A similar argument holds for all words less than B. Assume that S_7 contains a one corresponding to a word less than B. This situation could only exist after the execution of step (2) and would, therefore, mean that the word had not previously failed (or else a zero would be recorded in the corresponding position in S_8) and that when $b_i = 0$ there is a one in the word's ith bit position. Hence, the word would be greater than B, the desired contradiction.

The binary content of the words, A and B, determine the function formed by the Iterative Array. These two words are processed by an arithmetic computer [Fig. 1] which in turn gives the appropriate commands to the Iterative Array and the Transfer Memory. It is assumed that the time involved in this interrogation by the arithmetic computer is comparable to the time it takes for the execution of a command from the arithmetic computer by the Iterative Array or Transfer Memory. It is also assumed that interrogation and execution can be carried on simultaneously. Considering, then, only the time involved in the transfer of rows between the memory and the iterative processor and the time it takes to form a logic function, the calculation of the execution time for the between limits search is simple. Not including the retrieval of all words found in the search, the execution of the algorithm on 48 bit words takes 289 μsec. (See Appendix.)

Maximum and Minimum Search

The search for the minimum or maximum word stored in memory requires minor changes in the encoder. As mentioned on page 4 the encoder, in its present form, gives only a "SOME MARK" signal indicating the existence of at least one "0" in the M register. As a response to a minimum or maximum interrogation, it is desired to have three distinct signals with regard to

the number of "0"'s in the M register: (1) a "NO MARK" signal (simply the logical negation of the presently existing "SOME MARK" signal), (2) a "ONE MARK" signal (given when there exists only one "0" in the M register) and (3) a "PLURALITY" signal (given when there are more than one "0" in the M register). The existing equipment can easily be altered to give these responses.[1]

The words in the memory are divided into blocks of seven bits each. Starting at the most significant end of the words in memory, R_1 is transferred to S_7, R_2 to S_6, etc. Beginning with S_7, all registers are operated upon in the following manner.[2]

To Determine the Minimum Word

(1) Transfer to S_8 "0"'s corresponding to words being searched--all other positions contain "1"'s.

(2) Set N = 7

(3) $S_8 \vee S_n \longrightarrow M$

(a) If only one mark read out as the minimum word. Stop.

(b) If no marks go to step (4)

(c) If more than one mark:

$M \longrightarrow S_8$ go to step (4)

(4) Decrease n by 1

(a) If n = 0 transfer next section of the Transfer Memory to the Iterative Array and go to step (2).
If there are no more sections:

$S_8 \longrightarrow M$

Read out all words corresponding to "0" in M.

(b) If $n \neq 0$ go to step (3).

---

[1] Approximately 64 AND circuits in addition to existing logic.

[2] All commands, etc., would be furnished by an arithmetic computer as in the case of the between limits search.

To Determine the Maximum Word

(1) Transfer to S_8 "1"'s corresponding to words being searched--all other positions contain "0"'s

(2) Set n = 7

(3) $\overline{S_8 S_n} \longrightarrow M$

(a) If only one mark read out as maximum word. Stop.

(b) If no marks go to step (4)

(c) If more than one mark:

$\bar{M} \longrightarrow S_8$ go to step (4)

(4) Decrease n by 1

(a) If n = 0 transfer next section of the Transfer Memory to the Iterative Array and go to step (2)
If there are no more sections:

$\bar{S}_8 \longrightarrow M$ Read out all words corresponding to "0" in M.

(b) If $n \neq 0$ go to step (3)

Again the algorithms are duals--as would be expected after a similar development occurred in the previous section for the between limits search. Therefore, a discussion is given only for the minimum search.

Initially register S_8 contains "1"'s corresponding to words in the memory on which the search will not be performed. Such words are "masked" by "1"'s in S_8. Whenever a word is determined not to be a minimum it will also be masked by a "1" in S_8. Hence S_8 is said to contain the "mask" for the search. The logical sum of S_8 and S_n [step (3)] propagates the "1"'s contained in S_8 so that these positions will never cause a "ONE MARK" or a "PLURALITY" signal to appear. After the logical sum of S_8 and S_n has been taken one of three situations will be determined by the Encoder:

(1) There is only one "0" in M. This means that only one word has a "0" in row S_n (of those words still being checked) and is therefore, the minimum. [step (3a)]

(2) There is no "0" in M. This could only mean that all words being checked contain "1"'s in S_n and no information as to the minimum word can be determined. Therefore, the same mask (S_8) is used for the next row. [step (3b)]

(3) There is more than one "0" in M. This implies that more than one word and perhaps all words being checked have "0"'s in S_n. A further check is necessary to determine the minimum word. But a new mask may be needed because some of the words being tested up to this point could have "1"'s in this row and, of course, only words with "0"'s in M need be tested further. Therefore, M becomes the new mask. [step (3c)]

When the last row of the Transfer Memory has been checked an no minimum word has been found [step (4a)] then there is more than one minimum word and these words correspond to the "0"'s in S_8. When this occurs the contents of S_8 is transferred to M so that these words may be read out.

To show that this method correctly represents minimum words by "0"'s in S_8 (or M) assume the following cases:

(1) A word is the minimum word in the Transfer Memory and is being tested but in its corresponding position in S_8 there is a "1". Since S_8 is changed only at step (3c), this would mean the word has a "1" in S_n when other words have "0"'s. Hence, it is not a minimum word. This is the desired contradiction.

(2) A word is not a minimum but is being searched and at the end of the search its corresponding position in S_8 contains a "0". To be larger than the minimum word, a word must necessarily be "1" in at least one row where the minimum word is zero. At step (3c) a "1" would be transferred to S_8 in the position of the word in question--again the necessary contradiction.

Since the mask in S_8 propagates the information about words that have "failed" the search from one seven bit block to the next, there is no chance of a word being retrieved as a minimum word if it has been previously proven not a minimum.

The time it takes to complete a maximum or minimum search is within the bounds determined by the "worst case" and the "best case" situations. As a "worst case" situation the search would have to be conducted for all 48 rows of the Transfer Memory while a "best case" situation means the search is completed with the logical operation on the first row. Assuming 48 bit binary words, the "worst case" would take 170 μsec and the "best case" 15 μsec. (See Appendix.) Both the worst case and best case would take about 2000 μsec to execute with a standard [non-iterative] processor.

Ordered Retrieval

With very few changes, the minimum or maximum search algorithm may be utilized as an ordered retrieval algorithm. That is, all words may be retrieved from the memory in order corresponding to their arithmetic value. Because of the emphasis given the minimum search in the last section, this algorithm is used to construct an ordered retrieval algorithm by which words

are retrieved from memory starting at the minimum word and proceeding in order to the maximum word.

As in the case of the other searches, storing intermediate information obtained by this search in the Transfer Memory is to be avoided because of the time involved in transferring information between the memory and the processor. To accomplish this, register S_7 is used as a mask for all words that are not to be tested or have already been retrieved. Thus, when a word is retrieved as a minimum, a "1" is transferred to the corresponding position in S_7 causing the word to be masked for all later searches. S_8 will also contain a mask but it will function as in the minimum search--concerned only with finding a minimum word and not with which words have been retrieved.

Instead of dividing the words into blocks of seven bits, blocks of six bits are the maximum possible since S_7 is used as a mask. All specifications on the order of block transfers remain unchanged. Let it again be understood that all execution commands delivered to the Iterative Array and the Transfer Memory originate in the arithmetic computer. The following six steps constitute an ordered retrieval algorithm.

(1) Transfer to S_7 "0"'s corresponding to words being searched--all other positions contain "1"'s.

(2) $S_7 \longrightarrow S_8$

(3) Set N = 6

(4) $S_8 \vee S_n \longrightarrow M$

(a) If only one mark go to step (6)

(b) If no marks go to step (5)

(c) If more than one mark:

$M \longrightarrow S_8$, go to step (5)

(5) Decrease n by 1

(a) If n = 0 transfer next block to Iterative Array.

If there are no more blocks go to step (6)

(b) If n ≠ 0 go to step (4)

(6) $\bar{M} \vee S_7 \longrightarrow S_8$

$S_8 \longrightarrow S_7$

Read out words corresponding to "0"'s in M. Start with first bit block and repeat steps starting at step (3).

Notice that there is only one basic difference between this algorithm and the one for minimum search. This is step (6) which causes a new mask to be formed in S_7. All of the "0"'s in M are stored as "1"'s in S_7 (and S_8) because these zeros correspond to words to be retrieved. On the next search these positions will be effectively masked so no word will be retrieved more than once. With these positions masked, the search starts again at the most significant end of the words in the memory. This process continues until all words are retrieved (until S_7 contains only "1"'s).

With such a nondense set of words in the Transfer Memory it is difficult to set upper and lower bounds for the time it takes to complete an ordered retrieval. But as a pessimistic approximation of the "worst case" situation it is assumed that for each word retrieved the search must be performed on every row in the Transfer Memory. As a further approximation it is assumed that each row entails the formation of a logic function at step (4c). With these assumptions* the time for an ordered retrieval is about 180000 μsec. Taking into consideration the probability of two words in the Transfer Memory being equal (1 in 2^{48}) the "best case" choice could reasonably be the

---

* It is again assumed that the arithmetic computer speed need not be taken into consideration.

situation when all words are distinct and determined by the first ten rows of the Transfer Memory. Under this condition the time for an ordered retrieval execution is about 20000 μsec. (See Appendix.)

Seeber and Lindquist have proposed an associative memory system* capable of executing ordered retrieval [3]. In this system, the processing logic and storage elements are intermingled as opposed to the separation of the Iterative Array and Transfer Memory.

Instead of forming a logic function of the rows of the memory, this system uses an interrogation of each row, carried out by a device similar to a ternary counter. The three states of this counter are designated as "M", "0" and "1". When the counter is in the "M" state no interrogation of the row on which the counter is operating is required. When the counter is in the "0" state the row is searched for binary 0's and when it is in the "1" state a search is made for binary 1's. The cycle of the ternary counter is "M" to "1" to "0" and back to "M".

Connected to the memory is a Match Indicator (MI) which performs essentially the same function as the encoder in that it monitors the responses from the interrogation of the rows by the ternary counter and gives a no match (0), one match (1) or plurality match (P) signal corresponding to the result of the interrogation. Except at the start of the search, when all stages of the ternary counter are set to M, the setting of the ternary counter is a function of the Match Indicator signal.

Starting at the most significant end of the words in memory, the state of the first counter stage is changed from "M" to "0". An interrogation

---

*An associative memory is a memory in which words are located on the basis of at least part of their information content. The proposed memory uses cryotrons as storage cells. For a complete description of a cryotron memory see, R. R. Seeber, "Cryogenic Associative Memory," Nat. Conf. of the ACM, August, 1960.

is then made (in parallel) of all words in memory using 0MM ... M as the interrogation word. If the MI signal is "0" (indicating no matches to this interrogation) the state of the first counter stage is changed to "1" and another interrogation is tried. If the MI signal is "1", the word matching this interrogation word (0MM ... M) is retrieved and the state of the first counter stage is changed to "1". If the MI signal is "P" the state of the first stage is not changed but the state of the counter stage connected to row 2 is changed from "M" to "0". The interrogation is next conducted using 00M ... M.

In general, the procedure for changing counter stages mentioned above is carried on throughout all rows of the memory. In addition, when a stage is advanced from the "1" state to the "M" state, a carry is produced and directed to the counter stage connected to the row containing the next bit position greater (in arithmetic value) than the row connected to the advancing counter stage. This carry changes the state of the recipient stage. For example, if the counter settings were 01000lMM...M and if the interrogation using these settings caused the MI signal to be "0", the sixth counter stage from the left would be advanced to "M" and the fifth stage to "1". The new ternary counter settings would then be 0100lMMM...M.

The example on page 20 is given to clarify the rules for this ordered retrieval operation. The search is conducted on seven words, each six bits in length. The chart shows when each word is retrieved, the MI signal after each interrogation and the ternary counter settings for each interrogation.

Seeber and Lindquist consider a retrieval try as one interrogation of the memory. In the example given there are 17 retrieval tries. The ratio of number of retrieval tries to the number of successful retrievals is found by experimentation to be in the range from two to three for small random-sample

		INTERROGATION																
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
State of the Ternary Counter Bit Positions	1	M	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	2	M	M	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1
	3	M	M	M	M	0	0	0	1	M	0	1	1	1	1	1	1	1
	4	M	M	M	M	M	0	1	M	M	M	M	0	0	0	0	0	1
	5	M	M	M	M	M	M	M	M	M	M	M	M	0	1	1	1	M
	6	M	M	M	M	M	M	M	M	M	M	M	M	M	M	0	1	M
State of MI		P	1	P	P	P	1	1	0	P	0	P	P	1	P	1	1	1

Register Bit Positions of the Words Retrieved	1	0	1	1	1	1	1	
	2	1	0	1	1	0	1	1
	3	0	0	1	1	0	1	1
	4	0	0	0	1	1	0	0
	5	1	0	1	1	0	0	1
	6	0	0	1	1	1	1	0
Interrogation when Retrieved		(1)	(5)	(16)	(17)	(7)	(13)	(15)

Example 1.

sorts. Letting one interrogation by their system correspond to the formation of a new logic function in the ordered retrieval developed for the Iterative Array,* the retrieval ratio for the Iterative Array search is 48 for the "worst case" condition and 5 for the "best case."

The method by Seeber and Lindquist is more efficient than the one developed for the Iterative Array because their search does not start at the beginning of the memory for each retrieval. In the case of the ordered retrieval derived from the minimum search, it is necessary to start at the beginning of the memory for each retrieval because after a retrieval, a new logic function must be formed and this logic function will not necessarily contain all rows as variables. This is due to the fact that some rows will cause a no mark signal at step (4) of the algorithm (caused by all words searched at that time being equal to "1" in that row). However, the information about which rows are to be included and which are not has been determined at step (4). This information can be stored in the arithmetic computer and used as a basis for determining execution commands for the Transfer Memory and Iterative Array.

As an example of how the new logic function is created to continue the search after a retrieval, assume the logic function that creates a "one mark" situation in M is $S_7 \vee R_1 \vee R_2 \vee R_4 \vee R_5 \vee R_5 \vee R_6 \vee R_8 \vee R_9$. This function, before row (9) is made part of the logical sum, creates a "plurality" signal or a "one mark" signal in M. This is the logic function that should be formed to continue the retrieval. Rows R_3 and R_7 must be inhibited from becoming part of the function by the arithmetic computer.

---

* This is valid because the logical function is an interrogation of the words in the Transfer Memory.

Rows with subscripts greater than the subscript of any row included in the logical sum being used in interrogation must not be inhibited from being tested as a possible addition to the logical sum. This is because each retrieval necessitates the formation of new masks in S_7 and S_8, and rows that have been inhibited on previous retrieval tries may be made part of the logical sum because of the new masks.

The following search algorithm is comparable to the one developed by Seeber and Lindquist in that the search does not start at the beginning of the memory after each retrieval.

(1) Transfer to S_7 and S_8 "0"'s corresponding to words being searched--all other positions contain "1"'s.

(2) Set n = 1

(3) $S_8 \vee R_n \longrightarrow M$

(a) If only one mark go to step (5)

(b) If no marks go to step (4)

(c) If more than one mark $M \longrightarrow S_8$, got to step (4)

(4) Increase n by 1

(a) If n = 49,* go to step (5)

(b) If $n \neq 49$ go to step (3)

(5) $\bar{M} \vee S_7 \longrightarrow S_8$

$S_8 \longrightarrow S_7$

Read out all words corresponding to "0"'s in M.

$S_7 \longrightarrow M$

(a) If only one mark read out the word corresponding to this mark. Stop

---

* Assuming 48 bit words.

(b) If no marks Stop

(c) If more than one mark go to step (6)

(6) Decrease n by 1

(a) If n = 0 go to step (2)

(b) If n $\neq$ 0 go to step (7)

(7) $\bigvee_{K=1}^{n} R_K \vee S_7 \longrightarrow M$

(a) If only one mark go to step (5)

(b) If no marks go to step (6)

(c) If more than one mark:

$M \longrightarrow S_8$, go to step (4)

(d) If no function can be formed go to step (4)*

In step (7) the logical sum is taken over rows 1 through n of the Transfer Memory that are not inhibited by the arithmetic computer. If the first n rows are inhibited then the search continues at the $(n+1)^{th}$ row as a consequence of (7b). Because of the necessity of forming the logic function at step (7) this algorithm is written using the rows of the Transfer Memory as variables of the logic functions instead of using the Iterative Array register in which the rows are processed. This is done to clarify the way the search function is derived.

The algorithm does not indicate when it is necessary to inhibit a row of the Transfer Memory because this is a function performed by the arithmetic computer. The arithmetic computer determines which rows to inhibit from the encoder signal at step (3).

---

* Step (4a) will never occur at this point because this would mean that all rows give no information.

Steps (1) through (5) constitute the ordered retrieval derived from the minimum search. Steps (6) and (7) continue the search at the first uninhibited row preceding the row that determined the retrieval at step (5). If this function gives no information [step (7c)] then the search starts at the second uninhibited row, etc.

Comparison of the Two Systems

The dependence of the number of steps necessary to execute an ordered retrieval upon the word composition of the memory complicates a comparison of the search derived for execution by the iterative processor with the search proposed by Seeber and Lindquist. However, some conclusions can be drawn.

Many of the Iterative Array operations are executed before and after an interrogation of a row and have no counterpart in the execution of ordered retrieval by the associative memory-ternary counter system. These operations can be considered preparatory operations since they (a) prepare the mask registers for the next interrogation [steps (3c),(5) and (7)], (b) determine if the search is to be continued [step (5)] or (c) are concerned with memory accesses [steps (3) and (7)]. The changing of the mask may be avoided in many cases by discreet use of overlapping operations.* The operations at step (5) and memory accesses cannot be avoided. The latter operation is a consequence of having only eight registers to process as many as 48 rows for a retrieval. For words of six bits or less the number of registers is sufficient but in the case of longer words, the number of registers may make it necessary to transfer a row from memory each time it is

---

* For instance, at step (3), $S_8 \vee R_n$ could be read into M and S_6. If the "some mask" situation occurs the next logical sum is $S_6 \vee R_n$. This sum is then read into S8 and M, etc.

made part of the logical sum. A transfer between the Iterative Array and the Transfer Memory is the most time consuming operation performed in the search. These preparatory operations are the limiting factors in the method of serial search executed by the Iterative Array as compared to the parallel search executed by the Seeber and Lindquist system.

Since the memory access time is the most time consuming operation in the iterative processor search, the worst case comparison between the two systems will be discussed in terms of the number of rows which must be accessed for each retrieval. The situation causing the "worst case" occurs at step (7), under the conditions that none of the n rows are inhibited (n being determined at step (6)) and that for all n, the logical sum formed at step (7) generates a "no mark" situation in M. Under these conditions the search must form at least n logical sums before any word will be retrieved. These n sums are generated at step (7) and, therefore, there are $\sum_{i=1}^{n} K = \frac{(n)(n+1)}{2}$ rows transferred from the memory to the processor for these operations. The associative memory-ternary counter system will interrogate a maximum of 2n times to determine the same information under the stated conditions. The comparison of 2n interrogations by the Seeber and Lindquist system to the $\frac{(n)(n+1)}{2}$ memory accesses necessary in the case of the iterative processor is admittedly pessimistic. But it does indicate that for ordered retrieval using the Iterative Array, the number of bits per word must be limited for assurance of a reasonable number of memory accesses per retrieval.

Example 2, page 26 shows the execution of the ordered retrieval search by the Iterative Array on the same words retrieved in Example 1, page 20, by the associative memory-ternary counter system. Interrogation (16) of Example 2 is not executed with an interrogation function but is a

No.	INTERROGATION Function	Step Being Executed	Encoder Signal	Rows Inhibited
1	$R_1 \vee S_8$	3	1	None
2	$R_1 \vee S_8$	3	0	None
3	$R_2 \vee S_8$	3	P	R_1
4	$R_2 \vee R_3 \vee S_8$	3	P	R_1
5	$R_2 \vee R_3 \vee R_4 \vee S_8$	3	1	R_1
6	$R_2 \vee R_3 \vee S_7$	7	1	R_1
7	$R_2 \vee S_7$	7	0	R_1
8	$R_2 \vee S_8$	3	0	R_1
9	$R_3 \vee S_8$	3	0	R_1, R_2
10	$R_4 \vee S_8$	3	P	R_1, R_2, R_3
11	$R_4 \vee R_5 \vee S_8$	3	1	R_1, R_2, R_3
12	$R_4 \vee S_7$	7	P	R_1, R_2, R_3
13	$R_4 \vee R_5 \vee S_8$	3	0	R_1, R_2, R_3
14	$R_4 \vee R_6 \vee S_8$	3	1	R_1, R_2, R_3, R_5
15	$R_4 \vee S_7$	7	1	R_1, R_2, R_3, R_5
16	S_7	5	1	- - -

Register Bit Positions of the Words Retrieved	Order in Which Words are Retrieved: 1	2	6	7	3	4	5
1	0	1	1	1	1	1	1
2	1	0	1	1	0	1	1
3	0	0	1	1	0	1	1
4	0	0	0	1	1	0	0
5	1	0	1	1	0	0	1
6	0	0	1	1	1	1	0
Interrogation When Retrieved	(1)	(5)	(15)	(16)	(6)	(11)	(14)

Example 2.

consequence of step (5) of the algorithm. This example uses only six bit words, therefore, it is necessary to perform only six transfers from the memory to the Iterative Array. Seeber and Lindquist estimate an interrogation and response from the MI to take 7 μsec with their system. This means the retrieval of the seven words in Example 1 takes 119 μsec compared to approximately 56 μsec for the Iterative Array (See Appendix). If a row had to be read into the Iterative Array each time the row is made part of a logical sum, the time to execute the ordered retrieval by the Iterative Array is 88 μsec. It must be noted that the time comparison is not the best method of comparing the two systems. The interrogation and response time quoted by Seeber and Lindquist is not the optimum for their system.

Another approach to the problem of executing ordered retrieval using the Iterative Array is to program the ternary counter operation in the arithmetic computer. The arithmetic computer can then determine the interrogation function from the counter. For example, if the counter setting is 101001MM...M, the result of the logic operation $\overline{R}_1 \vee R_2 \vee \overline{R}_3 \vee R_4 \vee R_5 \vee \overline{R}_6$ stored in register M will create a zero in M corresponding in position to a word equal to 101001 in its first six bit positions.* From this information in the M register the encoder will signal the next setting of the counter, as did the Match Indicator in the system proposed by Seeber and Lindquist. However, the effectiveness of this method of ordered retrieval is also limited by the number of transfers necessary from the memory to the iterative processor.

---

*For a discussion of this method of search see [2].

CHAPTER IV

CONCLUSIONS

The three search algorithms derived for the iterative processor give an indication of the versatility of the processor. The between limits search and the minimum search are well adapted to the serial processing of the words in the Transfer Memory by the Iterative Array. This is true for the between limits search because the words determining the upper and lower bounds on the search are processed serially. The minimum search must be executed serially because the interrogation of the memory is done without previous knowledge of the memory content. The ordered retrieval search, when involving words of six bits or less, is also well executed by the processor, as is shown in Example 2.

The memory accesses necessary in the execution of a search by the Iterative Array constitute the greatest time factor in every search (see Appendix) and, therefore, is a limiting factor in the processor's application. However, it must be understood that the Iterative Array is designed for the purpose of pattern processing and the number of registers it contains (which determines the number of row transfers necessary) is designated to fulfill this purpose.

A comparison of the execution times of an ordered retrieval search on the same words (Examples 2 and 3) shows the search executed by the Iterative Array is faster than the search executed by the Seeber and Lindquist system. But the examples also show that the total number of operations necessary to execute the search with the associative memory-ternary counter system is less than the number necessary with the Iterative Array. In general this operation

difference will exist, which demonstrates the logical superiority of a system that processes information within the memory.

The Iterative Array has an advantage over the Seeber and Lindquist system for executing ordered retrieval in that a row is never interrogated more than once for each retrieval. But this advantage is overshadowed by the necessity of executing so many memory accesses for a retrieval. To eliminate memory access time the number of registers in the Iterative Array (other than the M register) must equal the number of bits in the words being searched plus two for masking pruposes. (If the mask changing procedure mentioned on page 24 is also used this entails three registers for masking purposes.) In effect, the additional registers eliminate the need for the Transfer Memory. An excellent extension of the present investigation would be to explore the practicality of a memory containing the Iterative Array logic.*

Neither the possible uses of the iterative processor nor the discussion of the logic system it possesses is exhausted by this paper. Memory search is only one of many uses for the existing logic. It has been shown [2] that the Iterative Array is capable of executing simple arithmetic operations. An extension of the present investigation would be to find the practical and theoretical limits of using the Iterative Array as an arithmetic unit. Another extension would be to find the usefulness of the Iterative Array for solving coding theory problems such as computation of Hamming distances.

---

* It should be noted, however, that the cost per bit of the Iterative Array is one to two orders of magnitude greater than the cost per bit of the Transfer Memory.

BIBLIOGRAPHY

[1] B. H. McCormick, "The Illinois Pattern Recognition Computer--ILLIAC III," IEEEE Trans. on Elec. Computers, vol. EC-12, pp. 791-813; December 1963.

[2] S. R. Ray and K. C. Smith, "Design of an Iterative Pattern Recognition Processor," Digital Computer Laboratory, University of Illinois, Report No. 161; April, 1964.

[3] R. R. Seeber and A. B. Lindquist, "Associative Memory with Ordered Retrieval," IBM JRD, 6, No. 1, pp. 126-136; January, 1962.

APPENDIX

Operation	Execution Time
Information transfer between the Transfer Memory and the Iterative Array	2 μsec
Encoder operation [monitoring of M register]	0.5 μsec
Iterative Array logical operation	0.5 μsec
Word readout or address given	2 μsec

Assumption:

The arithmetic computer operating time may be ignored.

Calculation of operation times (not including readout time except in the case of ordered retrieval)

Between Limits Search

Number of information transfers between the Iterative Array and Transfer Memory	96
Number of encoder operations	0
Number of Iterative Array logic operations	194

Total Time = 2(96)μsec + 0.5(194)μsec = 289 μsec.

Minimum and Maximum Search

Case I "Worst Case"

Number of information transfers between the Iterative Array and Transfer Memory	49
Number of encoder operations	48
Number of Iterative Array logic operations	96

Total Time = 2(49)μsec + 0.5(96 + 48)μsec = 170 μsec

Case II "Best Case"

Number of information transfers between the Iterative Array and Transfer Memory 7

Number of encoder operations 1

Number of Iterative Array logic operations 1

Total Time = 2(7) μsec + 0.5(1+1) μsec = 15 μsec

Ordered Retrieval

Case I "Worst Case"

Number of information transfers between the Iterative Array and Transfer Memory (48)(1024) = 49152

Number of encoder operations (48)(1024) = 49152

Number of Iterative Array logic operations (96)(1024) = 98304

Number of word readout or addresses 1024

Total Time = 2(49152 + 1024) μsec + 0.5(98304 + 49152) μsec ≐ 180000 μsec

Case II "Best Case"

Number of information transfers between the Iterative Array and Transfer Memory (5)(1024) = 5120

Number of encoder operations (5)(1024) = 5120

Number of Iterative Array Logic operations (10)(1024) = 10240

Number of word readout or addresses 1024

Total Time = 2(5120 + 1024) + 0.5(5120 + 10240)μsec ≐ 20000 μsec

Example II

Number of information transfers between the Iterative Array and Transfer Memory	6
Number of encoder operations	23
Number of Iterative Array logic operations	37
Number of word readout or addresses	7

Total Time = $2(6 + 7)\mu sec + 0.5(23 + 37)\mu sec = 56\ \mu sec$

UNIVERSITY OF ILLINOIS-URBANA
510.84 IL6R no. C002 no.160-170(1964
Report /
3 0112 088398166